Lecture 4

Defining Functions
Academic Integrity Quiz

• **Remember**: quiz about the course AI policy
  - Have posted grades for completed quizzes
  - Right now, missing ~130 enrolled students
  - If did not receive at least 9/10, take it again

• If you are not aware of the quiz
  - Go to [http://www.cs.cornell.edu/courses/cs11110/](http://www.cs.cornell.edu/courses/cs11110/)
  - Click **Academic Integrity** in side bar
  - Read and take quiz in CMS
Recall: Modules

• Modules provide extra functions, variables
  ▪ **Example**: math provides math.cos(), math.pi
  ▪ Access them with the `import` command
• Python provides a lot of them for us
• **This Lecture**: How to make modules
  ▪ Atom Editor to *make* a module
  ▪ Python to *use* the module

Two different programs
We Write Programs to Do Things

- Functions are the **key doers**

<table>
<thead>
<tr>
<th>Function Call</th>
<th>Function Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Command to <strong>do</strong> the function</td>
<td>- Defines what function <strong>does</strong></td>
</tr>
<tr>
<td>&gt;&gt;&gt; <code>plus(23)</code></td>
<td><code>def plus(n):</code></td>
</tr>
<tr>
<td>24</td>
<td><code>return n+1</code></td>
</tr>
<tr>
<td>&gt;&gt;&gt;</td>
<td></td>
</tr>
</tbody>
</table>

- **Parameter**: variable that is listed within the parentheses of a method header.
- **Argument**: a value to assign to the method parameter when it is called
We Write Programs to Do Things

• Functions are the **key doers**

**Function Call**
- Command to do the function

```
>>> plus(23)
24
>>> 
```

**Function Definition**
- Defines what function does

```
def plus(n):
    return n + 1
```

**Parameter**: variable that is listed within the parentheses of a method header.

**Argument**: a value to assign to the method parameter when it is called
We Write Programs to Do Things

- Functions are the **key doers**

**Function Call**
- Command to *do* the function

```python
>>> plus(23)
24
>>> 
```

**Function Definition**
- Defines what function *does*

```python
def plus(n):
  return n+1
```

- **Parameter**: variable that is listed within the parentheses of a method header.
- **Argument**: a value to assign to the method parameter when it is called
We Write Programs to Do Things

• Functions are the key doers

Function Call

• Command to do the function

>>> plus(23)
24

Argument to assign to n

Function Definition

• Defines what function does

def plus(n):
    return n+1

Parameter: variable that is listed within the parentheses of a method header.

Argument: a value to assign to the method parameter when it is called
Anatomy of a Function Definition

def plus(n):
    """Returns the number n+1
    Parameter n: number to add to
    Precondition: n is a number"
    
x = n+1
    return x

9/4/18
Defining Functions
Anatomy of a Function Definition

```
def plus(n):
    """Returns the number n+1
    Parameter n: number to add to
    Precondition: n is a number"
    x = n+1
    return x
```

- **def** plus(n): - *Function Header*
- """Returns the number n+1
  Parameter n: number to add to
  Precondition: n is a number"
- x = n+1 - *Statements to execute when called*
- return x - *Statements to execute when called*

Use vertical lines when you write Python on exams so we can see indentation.

The vertical line indicates indentation.
The **return** Statement

- **Format**: `return <expression>`
  - Used to evaluate *function call* (as an expression)
  - Also stops executing the function!
  - Any statements after a `return` are ignored

- **Example**: temperature converter function
  ```python
def to_centigrade(x):
    """Returns: x converted to centigrade""
    return 5*(x-32)/9.0
  ```
A More Complex Example

<table>
<thead>
<tr>
<th>Function Definition</th>
<th>Function Call</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>def foo(a, b):</code></td>
<td><code>&gt;&gt;&gt; x = 2</code></td>
</tr>
<tr>
<td>&quot;&quot;&quot;Return something&quot;&quot;&quot;</td>
<td><code>x ?</code></td>
</tr>
<tr>
<td>Param a: number</td>
<td><code>&gt;&gt;&gt; foo(3, 4)</code></td>
</tr>
<tr>
<td>Param b: number&quot;&quot;&quot;</td>
<td>What is in the box?</td>
</tr>
<tr>
<td>x = a</td>
<td></td>
</tr>
<tr>
<td>y = b</td>
<td></td>
</tr>
<tr>
<td><code>return x * y + y</code></td>
<td></td>
</tr>
</tbody>
</table>
A More Complex Example

Function Definition

```python
def foo(a, b):
    """Return something
    Param a: number
    Param b: number"
    x = a
    y = b
    return x*y+y
```

Function Call

```python
>>> x = 2
>>> foo(3, 4)
```

What is in the box?

A: 2
B: 3
C: 16
D: Nothing!
E: I do not know

9/4/18

Defining Functions
A More Complex Example

Function Definition

```python
def foo(a, b):
    """Return something
    Param a: number
    Param b: number"
    x = a
    y = b
    return x * y + y
```

Function Call

```plaintext
>>> x = 2
>>> foo(3, 4)
```

What is in the box?

- A: 2  CORRECT
- B: 3
- C: 16
- D: Nothing!
- E: I do not know
Understanding How Functions Work

- **Function Frame**: Representation of function call
- A **conceptual model** of Python

Draw parameters as variables (named boxes)

- Number of statement in the function body to execute next
- **Starts with 1**

```
function name
parameters
local variables (later in lecture)
```

instruction counter
Defining Functions

Definition:
```
def to_centigrade(x):
    return 5*(x-32)/9.0
```

Call: `to_centigrade(50.0)`
Example: to_centigrade(50.0)

1. Draw a frame for the call
2. Assign the argument value to the parameter (in frame)
3. Execute the function body
   - Look for variables in the frame
   - If not there, look for global variables with that name
4. Erase the frame for the call

```python
def to_centigrade(x):
    return 5*(x-32)/9.0
```
Example: \texttt{to\_centigrade(50.0)}

1. Draw a frame for the call
2. Assign the argument value to the parameter (in frame)
3. Execute the function body
   - Look for variables in the frame
   - If not there, look for global variables with that name
4. Erase the frame for the call

\begin{verbatim}
def to\_centigrade(x):
    return 5\*(x-32)/9.0
\end{verbatim}
Example: `to_centigrade(50.0)`

1. Draw a frame for the call
2. Assign the argument value to the parameter (in frame)
3. Execute the function body
   - Look for variables in the frame
   - If not there, look for global variables with that name
4. Erase the frame for the call

```python
def to_centigrade(x):
    return 5*(x-32)/9.0
```

Executing the return statement

```
50.0
```

```
10.0
```

The return terminates; no next line to execute
Example: to_centigrade(50.0)

1. Draw a frame for the call
2. Assign the argument value to the parameter (in frame)
3. Execute the function body
   - Look for variables in the frame
   - If not there, look for global variables with that name
4. Erase the frame for the call

```python
def to_centigrade(x):
    return 5*(x-32)/9.0
```

But don’t actually erase on an exam
The specification is a **lie**:

```python
def swap(a, b):
    """Swap global a & b"""
    tmp = a
    a = b
    b = tmp
```

```python
>>> a = 1
>>> b = 2
>>> swap(a, b)
```

Global Variables

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

Call Frame

```
```

```python
>>> a = 1
>>> b = 2
>>> swap(a, b)
```
The specification is a lie:

```python
def swap(a, b):
    """Swap global a & b"""
    tmp = a
    a = b
    b = tmp
```

```python
>>> a = 1
>>> b = 2
>>> swap(a, b)
```

Global Variables

- a
  - 1
- b
  - 2

Call Frame

- swap
  - 2
- a
  - 1
- b
  - 2
- tmp
  - 1

Call Frames vs. Global Variables
Call Frames vs. Global Variables

The specification is a lie:

```
def swap(a,b):
    """Swap global a & b""
    tmp = a
    a = b
    b = tmp
```

>>> a = 1
>>> b = 2
>>> swap(a,b)

Global Variables

Call Frame
Call Frames vs. Global Variables

The specification is a lie:

```python
def swap(a,b):
    """Swap global a & b""
    tmp = a
    a = b
    b = tmp
```

```python
>>> a = 1
>>> b = 2
>>> swap(a,b)
```

Global Variables

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

Call Frame

<table>
<thead>
<tr>
<th>swap</th>
</tr>
</thead>
<tbody>
<tr>
<td>a 2</td>
</tr>
<tr>
<td>b 1</td>
</tr>
<tr>
<td>tmp 1</td>
</tr>
</tbody>
</table>
Call Frames vs. Global Variables

The specification is a **lie**:

```python
def swap(a,b):
    '''Swap global a & b'''
    tmp = a
    a = b
    b = tmp
```

```python
>>> a = 1
>>> b = 2
>>> swap(a,b)
```

Global Variables

```
a 1  b 2
```

Call Frame

ERASE THE FRAME
**Exercise Time**

### Function Definition

```python
def foo(a, b):
    """Return something
    Param x: a number
    Param y: a number"
    x = a
    y = b
    return x * y + y
```

### Function Call

```python
>>> x = foo(3, 4)
```

What does the frame look like at the start?
Which One is Closest to Your Answer?

A: 

```
foo
```

```
a  3  b  4
```

```
x  3
```

B: 

```
foo
```

```
a  3  b  4
```

C: 

```
foo
```

```
a  3  b  4
```

```
x  3
```

D: 

```
foo
```

```
a  3  b  4
```

```
x  3  y
```

9/4/18  Defining Functions  26
Which One is Closest to Your Answer?

A:  
\[
\begin{array}{c c c c c}
& \text{foo} & \text{a} & 3 & \text{b} & 4 \\
\end{array}
\]

B:  
\[
\begin{array}{c c c c c}
& \text{foo} & \text{a} & 3 & \text{b} & 4 \\
\end{array}
\]

C:  
\[
\begin{array}{c c c c c}
& \text{foo} & \text{a} & 3 & \text{x} & 3 \\
\end{array}
\]

D:  
\[
\begin{array}{c c c c c}
& \text{foo} & \text{a} & 3 & \text{b} & 4 \\
\end{array}
\]

E:  
\[
\begin{array}{c c c c c}
& \text{foo} & \text{a} & 3 & \text{x} & 3 \\
\end{array}
\]

\[
\text{foo} \\
\text{a} & 3 & \text{b} & 4 \\
\text{x} & 3 & \text{y} & 4 \\
\]

9/4/18

Defining Functions
Exercise Time

Function Definition

```python
def foo(a, b):
    """Return something
    Param x: a number
    Param y: a number"
    x = a
    y = b
    return x*y+y
```

Function Call

```python
>>> x = foo(3, 4)
```

B:

```
foo
   a  3  b  4
```

9/4/18  Defining Functions  28
Exercise Time

Function Definition

```python
def foo(a, b):
    """Return something
    Param x: a number
    Param y: a number"
    x = a
    y = b
    return x*y + y
```

1. `x = a`
2. `y = b`
3. `return x*y + y`

Function Call

```python
>>> x = foo(3, 4)
```

B:

```python
foo
```

```
<table>
<thead>
<tr>
<th></th>
<th>3</th>
<th>b</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

What is the next step?
Which One is Closest to Your Answer?

A: 

B: 

C: 

D: 

9/4/18 
Defining Functions
Exercise Time

Function Definition

def foo(a,b):
    '''Return something
    Param x: a number
    Param y: a number'''
    x = a
    y = b
    return x*y+y

Function Call

>>> x = foo(3,4)

C:

```
foo
  a 3  b 4
  x 3
```

9/4/18
Exercise Time

Function Definition

```python
def foo(a,b):
    """Return something
    Param x: a number
    Param y: a number"
    x = a
    y = b
    return x*y+y
```

Function Call

```python
>>> x = foo(3,4)
```

What is the next step?
Which One is Closest to Your Answer?

A:

```
foo
```

```
<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

```

```
<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

```

RETURN

3

B:

```
foo
```

```
<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

```

```
<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

```

RETURN

3

C:

```
foo
```

```
<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

```

```
<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

```

RETURN

16

D:

```
foo
```

```
<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

```

```
<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

```

RETURN

ERASE THE FRAME

9/4/18

Defining Functions
def foo(a, b):
    """Return something
    Param x: a number
    Param y: a number"
    x = a
    y = b
    return x*y+y

>>> x = foo(3,4)

A:
Exercise Time

Function Definition

```python
def foo(a, b):
    """Return something
    Param x: a number
    Param y: a number"
    x = a
    y = b
    return x*y + y
```

Function Call

```python
>>> x = foo(3, 4)
```

A:

```
foo
a 3  b 4
x 3  y 4
```

What is the next step?
Which One is Closest to Your Answer?

A: foo

B: foo

C: foo

D: ERASE THE FRAME

9/4/18
Defining Functions
Exercise Time

Function Definition

```python
def foo(a, b):
    """Return something
    Param x: a number
    Param y: a number"
    
    x = a
    y = b
    return x*y+y
```

Function Call

```python
>>> x = foo(3, 4)
C:
```

```plaintext
foo
<table>
<thead>
<tr>
<th>a</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>3</td>
</tr>
<tr>
<td>y</td>
<td>4</td>
</tr>
<tr>
<td>RETURN</td>
<td>16</td>
</tr>
</tbody>
</table>
```
def foo(a,b):
    """Return something
    Param x: a number
    Param y: a number"
    x = a
    y = b
    return x*y+y

>>> x = foo(3,4)

C:

What is the next step?
Which One is Closest to Your Answer?

A: foo

B: ERASE THE FRAME

C: foo

D: ERASE THE FRAME

9/4/18 Defining Functions
Exercise Time

Function Definition

```python
def foo(a, b):
    """Return something
    Param x: a number
    Param y: a number"
    x = a
    y = b
    return x*y+y
```

Function Call

```python
>>> x = foo(3, 4)
D:
```

```
1  x = a
2  y = b
3  return x*y+y
```

ERASE THE FRAME
Exercise Time

Function Definition

def foo(a, b):
    """Return something
    Param x: a number
    Param y: a number""

1  x = a
2  y = b
3  return x*y+y

Function Call

>>> x = foo(3, 4)

D:

Variable in global space

16

ERASE THE FRAME
Visualizing Frames: The Python Tutor

```python
1 def max(x, y):
2     if x > y:
3         return x
4     return y
5
6 a = 1
7 b = 2
8 max(a, b)
```

Frames and Objects:
- **Global frame**
  - `max` function
- **Objects**
  - `a` with value 1
  - `b` with value 2
  - `max` with values `x: 1` and `y: 2`
Visualizing Frames: The Python Tutor

```
1 def max(x, y):
2     if x > y:
3         return x
4     return y
5
6 a = 1
7 b = 2
8 max(a, b)
```

- **Global Space**
- **Call Frame**
Visualizing Frames: The Python Tutor

Global Space

Call Frame

Variables from second lecture go in here
Visualizing Frames: The Python Tutor

1 def max(x,y):
2     if x > y:
3         return x
4     return y
5
6 a = 1
7 b = 2
8 max(a,b)

Missing line numbers!
Defining Functions

Visualizing Frames: The Python Tutor

Line number marked here (sort-of)

Missing line numbers!
Next Time: Text Processing